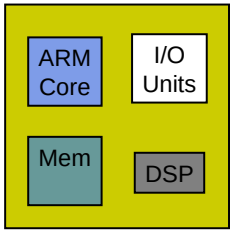
ARM Architecture and Instruction Set

Ingo Sander
ingo@imit.kth.se

ARM Microprocessor Core

- ARM is a family of RISC architectures, which share the same design principles and a common instruction set
- ARM does not manufacture the CPU itself, but licenses it to other manufacturers to integrate them into their own system
- The ARM Core as part of a system-on-chip



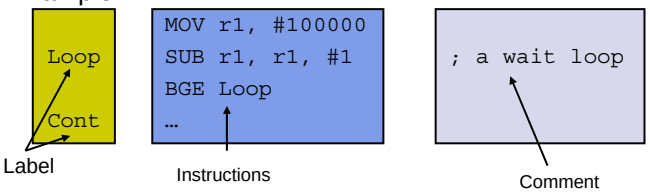
ASIC

ARM Microprocessor Core

- The ARM core is widely used in mobile phones, handheld organizers, and many other portable consumer devices
- Depending on the application ARM processors are available with e.g.
  - Different Cache Sizes
  - Different Bus Widths
  - Varying Clock Speeds
- Different Versions use different architectures, e.g.
  - ARM 7: von Neumann
  - ARM 9: Harvard
  - The assembly programs are not affected by the underlying architecture

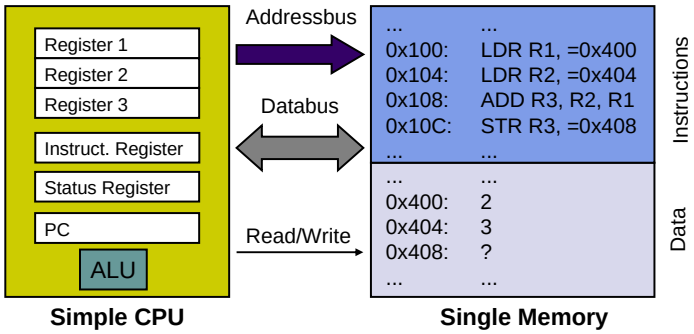
ARM assembly language

- The assembly language reflects the instruction set (almost one to one)
  - One instruction per line
  - Labels provide names for addresses (usually in first column)
  - Instructions often start in later columns.
  - Columns run to end of line
- Example:



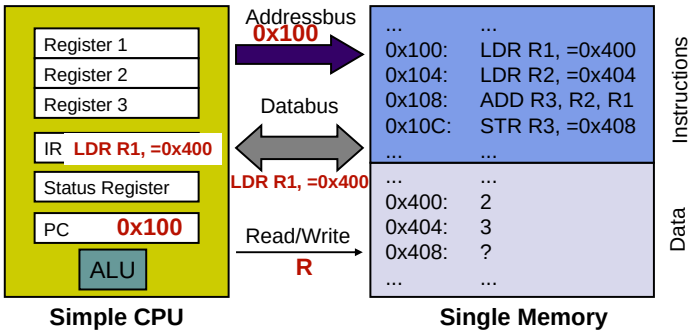
Von Neumann Architecture

- Consists of CPU and one single memory
- Memory holds instructions and data



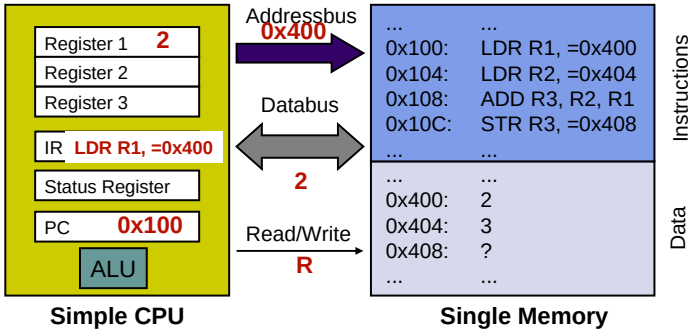
Example Von Neumann Architecture

- Start Address: 0x100
- Fetch Instruction



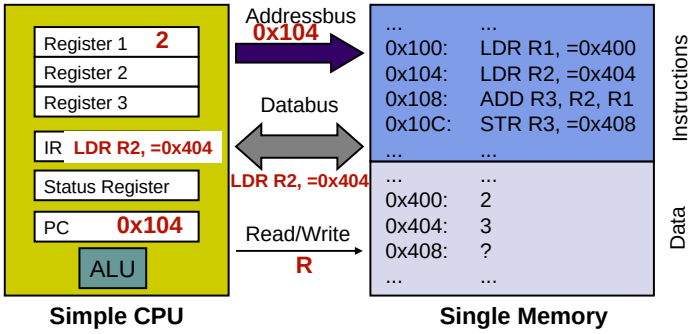
Example Von Neumann Architecture

- Execute Instruction



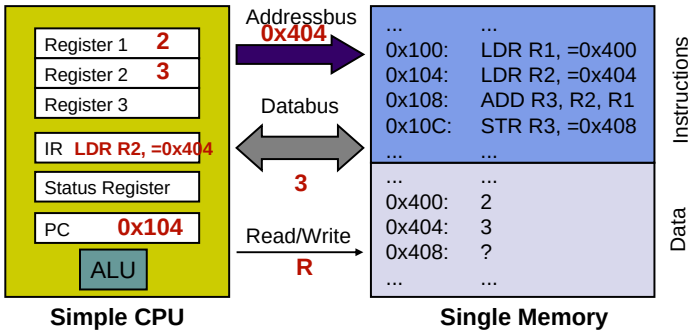
Example Von Neumann Architecture

- Increment Program Counter
- Fetch Instruction



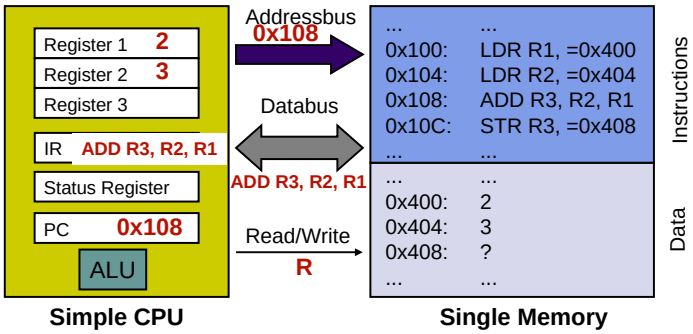
Example
Von Neumann Architecture

- Execute Instruction



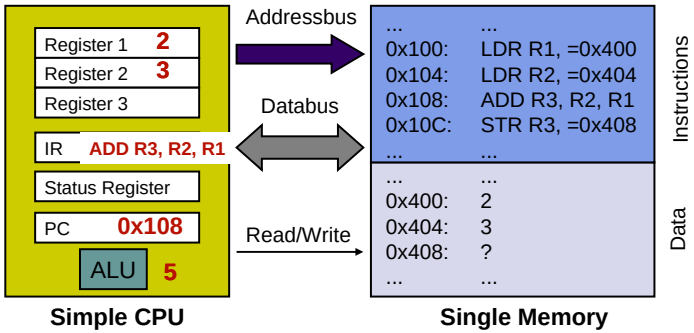
Example
Von Neumann Architecture

- Increment Program Counter
- Fetch Instruction



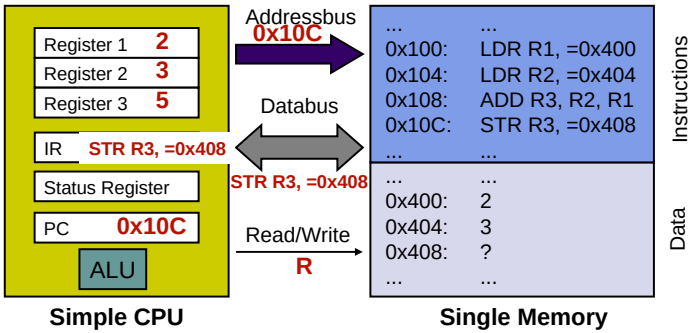
Example
Von Neumann Architecture

- Execute Instruction



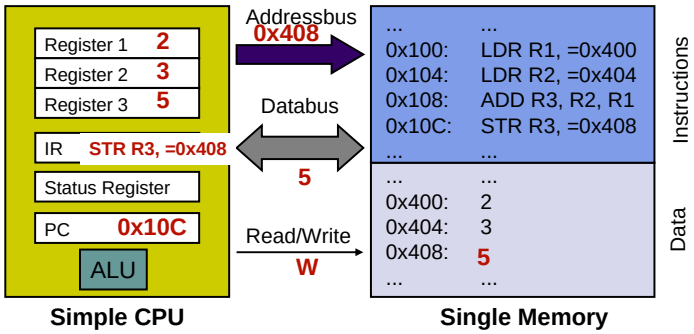
Example
Von Neumann Architecture

- Increment Program Counter
- Fetch Instruction



Example
Von Neumann Architecture

- Increment Program Counter
- Fetch Instruction

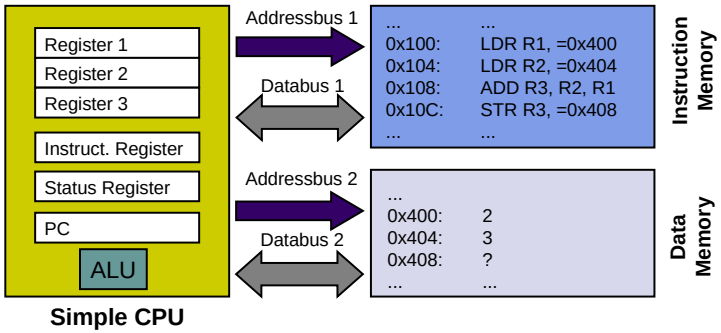


The von Neumann architecture

- Memory holds data, instructions.
- Central processing unit (CPU) fetches instructions from memory.
  - Separate CPU and memory distinguishes programmable computer.
- CPU registers help out: program counter (PC), instruction register (IR), general-purpose registers, etc.

Harvard Architecture

- Consists of CPU and two single memories
- In the original Harvard, one memory holds instructions and the other data

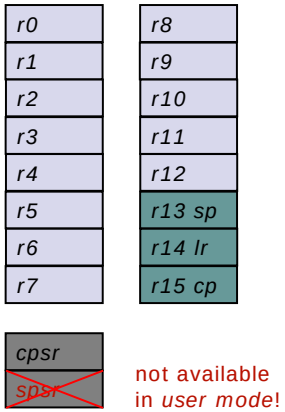


Comparison
von Neumann and Harvard

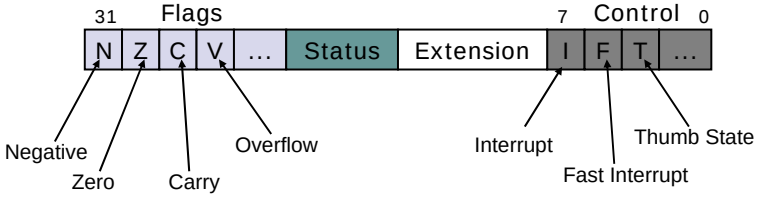
- Harvard allows two simultaneous memory fetches.
- Harvard can't use self-modifying code
- Most DSPs use Harvard architecture for streaming data:
  - greater memory bandwidth
  - more predictable bandwidth
- Additional hardware, since two address and data busses are needed

Programming Model: Registers available in *User Mode*

- The ARM processor has 17 active *registers* in *user mode*
  - 16 *data registers (r0-r15)*
  - 1 *processor status registers*
- The registers *r13-r15* have a special task
  - *r13* is the *stack pointer* (*sp*)
  - *r14* is the *link register* (*lr*)
  - *r15* is the *program counter* (*pc*)



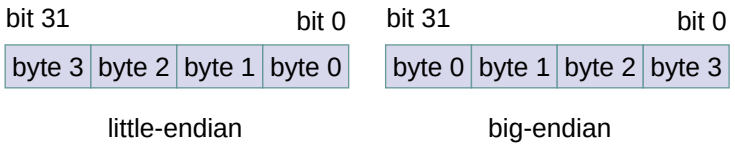
Generic Program Status Register



- The *cpsr* (Current Program Status Register) is used to monitor and control internal operations

Addresses and Endianness

- The ARM uses 32-bit addresses
- A Word is 32 bits (4 bytes) long
- An Address refers to a byte (not a word)
- The ARM processor can be configured to use a little-endian or big-endian memory system
  - Little-endian: lowest-order byte resides in the low-order bits of a word
  - Big-endian: lower-order byte resides in highest bits of the word



Data Movement

- The ARM has a *Load-Store architecture*
- Data operands must be loaded into registers before they can be processed by an ALU
- Data is moved between registers by means of Move instructions

```
MOV r1, r2    ; r1 = r2
MOV r3, #1    ; r3 = 1
```

- Data is moved between memories by Load and Store Instructions

Single Register-Memory Transfers

- Data operands must be loaded into registers before they can be processed by an ALU
- The Load and Store instructions can be combined with different *addressing modes*
- The basic *Load instruction* is LDR (load word into register), but there are variations that work on byte (LDRB), halfword (LDRH) and signed bytes (LDRSB)
- The basic *Store instruction* is STR (save word from a register), variations are STRB och STRH

Example for Load

Before:

```
r0 = 0x00000000
r1 = 0x00070000
mem32[0x00070000] = 0x00000005

LDR r0, [r1]
```

After:

```
r0 = 0x00000005
r1 = 0x00070000
```


Useful addressing modes: Preindexing

Before:

```
r0 = 0x00000000
r1 = 0x00007000
mem32[0x00007000] = 0x00001000
mem32[0x00007004] = 0x00002000
```

Preindexing: `LDR r0, [r1, #4]`

After:

```
r0 = 0x00002000
r1 = 0x00007000
```


Useful addressing modes: Preindexing with Writeback

Before:

```
r0 = 0x00000000
r1 = 0x00007000
mem32[0x00007000] = 0x00001000
mem32[0x00007004] = 0x00002000
```

Preindexing with Writeback: `LDR r0, [r1, #4]!`

After:

```
r0 = 0x00002000
r1 = 0x00007004
```

Useful addressing modes: Postindexing

Before:

```
r0 = 0x00000000
r1 = 0x00007000
mem32[0x00007000] = 0x00001000
mem32[0x00007004] = 0x00002000
```

Postindexing: `LDR r0, [r1], #4`

After:

```
r0 = 0x00001000
r1 = 0x00007004
```


Multiple Register-Memory Transfers

- Load-store multiple instructions are used to transfer multiple registers between memory and processor in a single instruction
  - LDM (Load Multiple Registers)
  - STM (Save Multiple Registers)
- There are four addressing modes: IA (increment after), IB (increment before), DA (decrement after), DB (decrement before)
- Be careful, which addressing mode you select, otherwise you may produce self-modifying code!

Example Load Store Multiple Instructions

Before:

```
r0 = 0x00000005
r1 = 0x00000006
r2 = 0x00000007
r3 = 0x00007000
```

```
STMIA r3!, {r0-r2}
MOV r0,#1
MOV r1,#2
MOV r2,#3
```

After (1):

```
mem32[0x00007000] = 0x00000005
mem32[0x00007004] = 0x00000006
mem32[0x00007008] = 0x00000007
r3 = 0x0000700C
```

```
LDMDB r3!, {r0-r2}
```

After (2):

```
r0 = 0x00000005
r1 = 0x00000006
r2 = 0x00000007
r3 = 0x00007000
```

- Such pairs of Load-Store Multiple Instructions can be used to temporarily store registers on the memory.

Stack Operations

- The ARM architecture uses load-store multiple instructions to *pop* and *push* data from and to the stack
- Here you have to decide, if the stack is ascending (A) or descending (D) and you use a full (F) or empty (E) stack.
  - Full Stack: Stack Pointer points at last used address
  - Empty Stack: Stack Pointer points at first empty address
- `STMFA sp!, {r5,r7}` pushes registers `r5` and `r7` on an ascending stack and points after the instruction on the memory location where `r7` is stored!
- `STMFA sp!, {r5,r7}` is equivalent to `STMIB r13, {r5,r7}`

Loading Constants

- There are two pseudo-instructions to load constants

```
LDR r1, =0x7000 ; loads r1 with constant 0x7000
ADR r2, label   ; loads r2 with address for label
```


Data Processing Instructions

- Data processing instructions manipulate data within registers (Move, Arithmetic, Logical, Comparison, Multiply)
- If the S suffix is used the CPSR flags N, Z, C, V are updated
  - ADD r1, r2, r3 does not update CPSR
  - ADDS r1, r2, r3 updates the CPSR

Data Processing Instructions and CPSR

```
MOVS r1, #1
⇒ NZCV = 0000
MOVS r2, #-1
⇒ NZCV = 1000
ADD r3, r2, r1
⇒ NZCV = 1000
ADDS r3, r2, r1
⇒ NZCV = 0110
```


Data Processing Instructions

- Move: MOV, MVN
- Arithmetic: ADD, ADC, SUB, SBC, RSB, RSC
- Logical: AND, ORR, EOR, BIC
- Comparison: CMP, CMN, TST, TEQ
- Multiply: MUL, MLA, SMLAL, SMULL, UMULL, SMLAL, UMLAL

Formats for data processing instructions

- Basic format

```
SUB r3, r2, r1     ; r3 = r2 – r1
```

- Immediate Operand

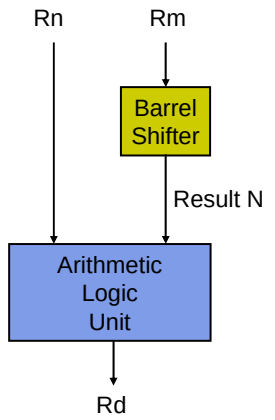
```
SUB r3, r2, #3     ; r3 = r2 - 3
```

- Preprocessing (Barrel-Shifter)

```
SUB r3, r2, r1, LSL #1 ; r3 = r2 – (r1 * 2)
```


The Barrel Shifter

- The barrel shifter allows an initial shift operation before it enters the ALU
- Shift Operations: `LSL, LSR, ASR, ROR, RRX`
- Example:
- `MOV r3, r4, LSL #3 ;r3 = 8 * r4`



Branch Instructions

- The branch instruction is used to change the flow of execution (if-then-else, for-loop, while-loop)
- Branch Instructions: `B, BL, BX, BLX`
- Branches are often used with conditions (`EQ, NE, CS, CC, MI, PL, VS, HI, LS, GE, LT, GT, LE`)
  - `BEQ label` ; Branch to label, if `Z = 1`
- The address `label` is stored in the instruction as a PC-relative offset and must be within 32MB of the branch instruction

Subroutines

- The BL (Branch and Link) instruction can be used for subroutines, since it writes the returnaddress to the link register

```
    BL subroutine
    ...
subroutine
    ...             ; code for subroutine
    MOV pc, lr      ; return by moving lr to pc
```

Conditional Execution

- Not only branch instructions can be used with conditions
  - `ADDEQ r4, r5, r6` is only executed if `Z = 1`
- Conditional Execution helps to design shorter programs that do not use so much memory

Summary

- ARM is a family of microprocessor cores
- Load/store architecture
- Most instructions are RISCy, operate in single cycle
  - Some multi-register operations take longer
- All instructions can be executed conditionally